Introduction to Operating Systems

What's an OS?

The OS is a layer between applications and hardware to ease development.

- **Abstraction**. provides abstraction for applications:
  - manages + hides hardware details
  - uses low-level interfaces (not available to applications)
  - multiplexes hardware to multiple programs (*virtualization*)
  - makes hardware use efficient for applications
- **Protection**.
  - from processes using up all resources (*accounting, allocation*)
  - from processes writing into other processes memory
- **Resource Management**.
  - manages + multiplexes hardware resources
  - decides between conflicting requests for resource use
  - *goal*: efficient + fair resource use
- **Control**.
  - controls program execution
  - prevents errors and improper computer use
- ⇝ **no universally accepted definition**

Hardware Overview

- **Bus**: CPU(s)/devices/memory (conceptually) connected to common bus
  - CPU(s)/devices competing for memory cycles/bus
  - all entities run concurrently
  - *today*: multiple buses
- **Device controller**: has local buffer and is in charge of particular device
- **Interplay**:
  1. CPU issues commands, moves data to devices
  2. Device controller informs APIC that it has finished operation
  3. APIC signals CPU
  4. CPU receives device/interrupt number from APIC, executes handler

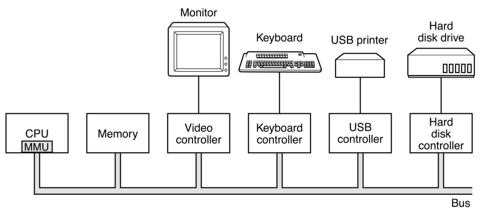

Figure 1: Traditional bus design.

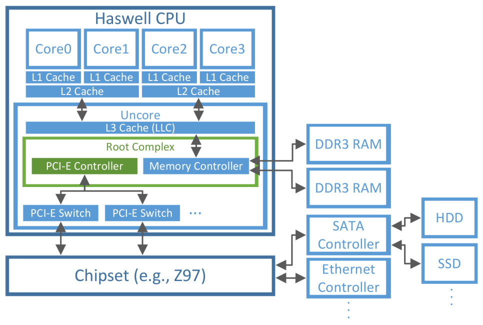

Figure 2: Modern bus design.

Central Processing Unit (CPU) — Operation

- **Principle**:
  1. *fetches* instructions from memory,
  2. *executes* them
- **During execution**: (meta-)data is stored in CPU-internal registers, i.e.
  - general purpose registers
  - floating point registers
  - instruction pointer (IP)
  - stack pointer (SP)
  - program status word (PSW)

CPU — Modes of Execution

- **User mode** (x86: *Ring 3/CPL 3*):
  - only non-privileged instructions may be executed
  - cannot manage hardware → *protection*
- **Kernel mode** (x86: *Ring 0/CPL 0*):
  - all instructions allowed
  - can manage hardware with *privileged instructions*

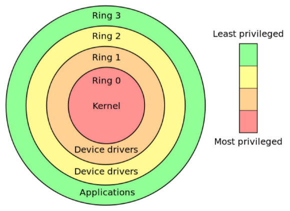

Figure 3: The different protection layers in the *ring model*.

Random Access Memory (RAM)

- **Principle**: keeps currently executed instructions + data
- **Connectivity**:
  - *today*: CPUs have built-in **memory controller**
  - *CPU caches*: "wired" to CPU
  - *RAM*: connected via pins
  - *PCI-E switches*: connected via pins

Caches

- **Problem**: RAM delivers instructions/data slower than CPU can execute
- **Locality principle**:
  - *spatial locality*: future refs often near previous accesses (e.g. next byte in array)
  - *temporal locality*: future refs often at previously accessed ref (e.g. loop counter)
- **Solution**: *caching* helps mitigating this memory wall
  1. *copy* used information temporarily from slower to faster storage
  2. *check* faster storage first before going down *memory hierarchy*
  3. if *not found*, data is copied to cache and used from there
- **Access latency**:
  - *register*: ~1 CPU cycle
  - *L1 cache* (per core): ~4 CPU cycles
  - *L2 cache* (per core pair): ~12 CPU cycles
  - *L3 cache/LLC* (per uncore): ~28 CPU cycles (~25 GiB/s)
  - *DDR3-12800U RAM*: ~28 CPU cycles + ~ 50ns (~12 GiB/s)

Device controlling

- **Device controller**: controls device, accepts commands from OS via *device driver*
- **Device registers/memory**:
  - *control* device by writing device registers
  - *read* status of device by reading device registers
  - *pass data* to device by reading/writing device memory
- **Device registers/memory access**:
  1. **port-mapped IO** (PMIO): use special CPU instructions to access port-mapped registers/memory
  2. **memory-mapped IO** (MMIO):
     - use same address space for RAM and device memory
     - some addresses map to RAM, others to different devices
     - access device's memory region to access device registers/memory
  3. **Hybrid**: some devices use hybrid approaches using both

Summary

- The OS is an abstraction layer between applications and hardware (multiplexes hardware, hides hardware details, provides protection between processes/users)
- The CPU provides a separation of User and Kernel mode (which are required for an OS to provide protection between applications)
- CPU can execute commands faster than memory can deliver instructions/data — memory hierarchy mitigates this memory wall, needs to be carefully managed by OS to minimize slowdowns
- device drivers control hardware devices through PMIO/MMIO
- Devices can signal the CPU (and through the CPU notify the OS) through interrupts